REMARKS

Status of the Claims

- Claims 1-24 are pending in the Application after entry of this amendment.
- Claims 1-24 are rejected by Examiner.

Claim Rejections Pursuant to 35 U.S.C. §103 (a)

Claims 1-23 stand rejected pursuant to 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 5,187,786 to Densmore et al. in view of U.S. Publication No. US 2004/0133882 to Angel et al. Applicant respectfully traverses the rejection.

Claim 1 recites:

> A method of locating classes in a class path, the method comprising:
> generating a cache of information relating to the classes in the class path;
> creating a wrapper for selected elements in the class path to provide a level of indirection from application programming interfaces used by a class locator, the wrapper indirection level providing for different caches to be used for the selected elements;
> requesting a search of the class path via the wrapper; and
> searching the cache to satisfy the requested search.

Densmore et al. teaches a method and apparatus for implementing a class hierarchy of objects in a hierarchical file system which does not require the support of additional file attributes by the hierarchical file system, and has particular application to object oriented programming for a window-based computer systems. Each object is organized as a class instance of a class, or a class. These classes and a root class are organized into the class hierarchy. (col. 2 lines 46-54).

Densmore at col. 4 lines 60-67 also states:

> "The hierarchy of a root class directory and root class files 20 implements the root class of the class hierarchy of objects. The hierarchy 20 comprises a root class directory 22, a root class path file 24 and a class instance making method file 26. The root class directory 22, the root class path file 24 and the class instance making method file 26 may be stored on any storage subsystem of a computer system, preferably on a direct access storage subsystem." (col. 4, lines 60-67).

Applicant submits that although a storage system is taught in Densmore et al., different caches for selected elements in the class path as recited in Claim 1 are not specifically taught.

Further, Densmore et al. teaches at col. 5, lines 6-14:

> "The root class path file 24 facilitates invocation of the root class method. The root class path file 24 is named with a special name indicative of the nature of the file, for example, "PATH". The root class path file 24 comprises a plurality of class path directory names. The class path directory names are accessed by the class making procedure 12 when implementing a new class subclass to the root class, and by the message sending procedure 14 when invoking the root class method." (col. 5, lines 6-14).

Densmore et al. teaches accessing the class path directory names. Applicant submits that Densmore et al. does not teach satisfying the requested search by searching the cache which contains selected elements of the class path. Applicant submits that Densmore et al. doses not explicitly teach or suggest selecting elements in the class path and providing different caches for those elements and then using the caches to satisfy the requested search as indicated by Claim 1.

Applicant agrees with the Examiner that Densmore et al. does not explicitly teach a level of indirection from application programming interfaces used by a class locator, the wrapper indirection level providing for different caches to be used for the selected elements. However, Applicant respectfully disagrees that Angel et al. comprehensively teaches the missing elements.

Angel et al. teaches instrumenting a computer program to provide instrumented byte code which includes examining the byte code, selecting portions of the byte code for instrumentation, and instrumenting the portions to provide instrumented byte code. Selecting the portions may include choosing portions of the byte code corresponding to method entry, method exit, a throw, a method call, or a new line number. Instrumenting a portion of the byte code corresponding to a method call may include instrumenting a local line number of source code corresponding to the byte code being instrumented. Instrumenting the portions may include adding calls to instrumentation runtime functions that pass parameters indicative of the portions being instrumented. (Abstract).

Angel et al. explains that:

> "Code instrumentation is performed by adding statements to software in order to monitor performance and operation of the software during run time. Code instrumentation is sometimes used to facilitate debugging of run time errors relating to memory accesses. Specifically, since many run time errors are the result of improperly accessing or using memory (e.g., writing beyond an array's boundaries, not freeing dynamically allocated memory, etc.), then instrumentation may be used to supplement memory accessing portions of the software with additional software that monitors memory accesses and provides an indication when it appears that an improper access has occurred." (paragraph 0005).

Applicant submits that Claim 1 does not recite any instrumentation, or monitoring, or augmentation of byte code, or augmentation of any other type of code. The Examiner points to paragraph 0016 to address the "wrapper" that provides a level of indirection that provides for different caches as recited in Claim 1. Yet paragraph 0016 of Angel et al. states:

"A method may be instrumented to provide instrumentation for handling an abort. A native function call may be instrumented by adding a byte code wrapper to the native function and then instrumenting the wrapper. The wrapper may include byte code corresponding to method entry and exit portions. Instrumenting a call to a native function may include providing an native assembly language thunk that captures data passed to and from the native function. The assembly language thunk may be hooked between the virtual machine and the call to the native function. Hooking the assembly language thunk may include intercepting a call that provides an address for a procedure." (paragraph 0016)

Applicant submits that Angel et al. seems to only uses a byte code wrapper to instrument native code. Applicant also notes that Claim 1 does not include instrumentation or byte code in conjunction with a wrapper. In addition, Angel et al. states in paragraph 0169:

> "If it is determined at the test step 610 that the method is implemented in native code (by examining the access_flags in the class), control passes from the step 610 to a step 612 where instrumenting the native code is handled by, for example, adding a byte code wrapper to the method. The wrapper causes the VM (and the instrumentation software) to treat the native method as a conventional byte code method." (paragraph 0169).

Applicant submits that the "wrapper" of Angel is used in association with a byte code to instrument native code to cause a Virtual Machine and the associated instrumentation software to execute the native method as a conventional byte code method. Applicant submits that Claim 1 does not recite a need to instrument native code for a virtual machine, or to use instrumentation software, or a to use conventional byte code instead of native code as

specifically taught by Angel et al. Angel et al. does not teach using a wrapper as a level of indirection providing from APIs used by a class locator for different caches as recited in Claim 1. Angel et al. is silent on using a wrapper for any other purpose other than that provided in paragraphs 0016 and 0169 above.

Accordingly, Applicant suggests that one of skill in the art would not use the teaching of Angel et al. of a wrapper as a function to cause a virtual machine associated with instrumentation code and software to use conventional byte code instead of native code in conjunction with the class hierarchy implementation teaching of Densmore et al. to arrive at the recitation of Claim 1, where wrappers provide a level of indirection to APIs for different caches in conjunction with requesting and satisfying a search of a class path.

There is no suggestion in Densmore et al that a wrapper be used at all. Inserting a wrapper that provides a level of indirection to APIs in conducting a search of a class path is not taught or suggested in Densmore et al. Applicant is unsure if the invention of Densmore et al. would continue to operate properly if combined with Angel et al. Similarly, Angel et al. does not teach or suggest that a wrapper can be used to provide a level of indirection to APIs for different caches used in the satisfaction of a class search request. Instead Angel et al. teaches that a wrapper is useful for instrumenting native code, which is not a function recited in Claim 1. As a result, one of skill in the art would not have been inclined to use the combine the two references to achieve the result of Claim 1. Accordingly, Applicant respectfully suggests that the motivation to combine does not lie in the documents themselves and does not lie in one of skill in the art because the combination of the two references does not produce the functionality of Claim 1.

In addition, looking at the claim as a whole, the combination of Densmore et al. and Angel et al., if combined, does not genuinely and coherently teach, in an integrated fashion, generating a cache of information, creating a wrapper for selected elements in the class path to provide a level of indirection used by a class locator where different caches are used for the selected elements, then requesting a search of the class path via the wrapper; and searching the cache in order to satisfy the search request.

Applicant respectfully submits that the combination of Densmore et al and Angel et al., even if combined, would not teach or suggest the specific recitation in Claim 1. Applicants submit that if, for the sake of argument, that Densmore et al. and Angel et al. were

combined, the result might be a Densmore et al. hierarchy which incorporates instrumentation for monitoring purposes. Applicant notes that this is not the subject mater or the recitation of Claim 1.

Since similar aspects of a wrapper providing a level of indirection and multiple caches are present in independent Claims 10, 17, 22, and 23, then Applicant respectfully traverses the current 35 U.S.C §103(a) rejection for the above stated reasons and submits that Claims 1-23 patentably define over the cited art. Accordingly, Applicant respectfully requests reconsider and withdraw of all Claim 1-23 rejections under 35 U.S.C. §103(a).

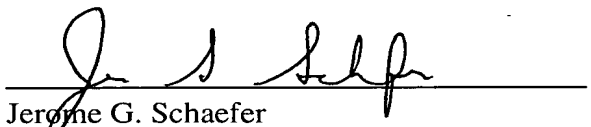
Claim 24 stands rejected pursuant to 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 5,187,786 to Densmore et al. in view of U.S. Publication No. US 2004/0133882 to Angel et al. in further view of Harter et al. Applicant respectfully traverses the rejection based on the arguments above and that Harter et al. also fails to teach at least the elements of a level of indirection from application programming interfaces used by a class locator, the wrapper indirection level providing for different caches to be used for the selected elements. Therefore Claim 24 cannot be rendered obvious by the combination of Densmore et al, Angel et al., and Harter et al. Accordingly, Applicant respectfully requests withdraw the Claim 24 rejection under 35 U.S.C. §103(a) as it patentably defines over the cited art.

Conclusion

Applicant respectfully request reconsideration of the subject application in light of the remarks presented above. A Notice of Allowance for all pending claims is earnestly solicited.

Respectfully Submitted,

Date: June 15, 2006



Jerome G. Schaefer
Registration No. 50,900

Woodcock Washburn LLP
One Liberty Place - 46th Floor
Philadelphia PA 19103
Telephone: (215) 568-3100
Facsimile: (215) 568-3439